VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Optimalizace nákladky na kamiony ve firmě Gates Hydraulics s.r.o.

Optimization of Trucks Loading in Gates Hydraulics s.r.o.

2014 Jakub Bílý

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Jakub Bílý**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Optimalizace nákladky na kamiony ve firmě Gates Hydraulics s.r.o.
Optimization of Trucks Loading in Gates Hydraulics s.r.o.

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit vhodné řešení optimálního naložení nákladu do kamionu pro firmu Gates Hydraulics s.r.o. V rámci práce bude jako základ využit jeden ze standardních algoritmů řešící tento problém (např. CLP nebo BPP). Vybraný algoritmus bude upraven podle požadavku firmy a bude provedena vizualizace rozmístění nákladu do kamionu.

Jednotlivé cíle práce:
1. Nastudovat možnosti jednotlivých algoritmů poskytujících řešení problému.
2. Návrh a implementace vhodného algoritmu pro řešení problému.
3. Návrh a implementace vizualizace výstupů algoritmů.
4. Provedení testů a demonstrace výsledků.

Seznam doporučené odborné literatury:

[1] S. Martello, D. Pisinger, D. Vigo: The Three-Dimensional Bin Packing Problem, Operations Research 2000
[2] F. Parreno, R. Alvarez-Valdes, J. M. Tamarit, J. F. Oliveira: A Maximal-Space Algorithm for the Container Loading Problem. INFORMS J. on Computing 20, 3 (July 2008), 412-422

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Student: Jakub Bílý

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Zveřejněna bude pouze veřejná část práce, která bude poskytnuta dle výše uvedených požadavků. Neveřejná část práce bude k dispozici pouze oponentovi práce a komisi pro potřeby obhajoby práce. Po obhajobě bude neveřejná část práce studentovi vrácena.

Student bude s neveřejnou částí práce nakládat v souladu s prohlášením v Rámcové dohodě o mlčenlivosti

V Karviné 5. května 2014

Ing. Anton Svrček, Gates Hydraulics s.r.o.

Student: Jakub Bílý

Požadavek na report vzešel od vedoucího skladu. Je logickým doplňkem dalších logistických nástrojů, které jsme ve firmě vyvinuli pro potřeby řízení plynulého toku materiálu k zákazníkovi.

Počáteční představa byla jasná: Přehledný, minimalistický řešený, report zobrazující požadované objemy na kamionovou přepravu zboží k jednotlivým zákazníkům. Report nahrazující desítky minut trvající monotónní výpočetní postup realizovaný v excelu.

Úlohu řešili 2 studenti, kteří zpracovali analýzu, design a implementaci. Vývoj probíhal agilně a uvítali jsme, že na většině setkání byli společně. Nejprve nám předvedli funkční algoritmus realizující úlohu uložení zboží na palety. Ten je základem celého reportu. Ocenili jsme možnost grafického zobrazení uloženého zboží, nicméně jsme dále trvali na minimalistickém výstupu, který má pro odpovědného pracovníka nejvyšší hodnotu.

Poté jsme se dohodli na datovém rozhraní a rozdělili zodpovědnosti za data. Před vlastní integrací do našich systémů studenti doladili grafický design a navrhli rozhraní pro konfiguraci obecných parametrů a parametrů jednotlivých zákazníků.

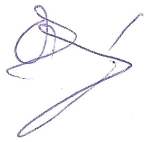
V Karviné 5.5.2014

Ing. Anton Svrček, Gates Hydraulics s.r.o.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Jakub Bílý

V Ostravě, 7.5.2014



Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli. Jmenovitě pan Ing. Jan Martinovič, Ph.D. za cenné rady, trpělivost a odborné vedení. Panu Antonu Svrčkovi z firmy Gates Hydraulics s.r.o. za jeho přístup a možnost podílet se na takto zajímavém projektu.

Abstrakt

Tato práce se zabývá problematikou naložení nákladu na kamion v co možná nejefektivnější formě, tedy rychle a přesně. Řešíme zde problém uložení co největšího počtu objektů (beden) na co nejmenší počet palet, které jsou následně transportovány kamiony. Kdysi bylo nutné spolehnout se jen na dobrý odhad, ale dnes můžeme tuto problematiku díky počítačům řešit mnohem efektivněji. Moderní doba vyžaduje rychlost, efektivitu a nízké ceny. Toho se snaží dosáhnout i všechny malé i velké transportní firmy. Díky tomu se i počítačová věda již několik let tomuto odvětví věnuje a snaží se přijít se stále lepšími metodami, jak problém vyřešit. Základ řešení našeho problému vychází z předešlé studie autorů David Pisinger, Silvano Martello, Daniele Vigo [4] z roku 1997, kde popsali problém zvaný Binpacking, což je jedna z metod řešení optimalizace nákladu. Tato bakalářská práce je psána pomocí sázecího systému LaTeXa implementace je napsána v programovacím jazyce C# v prostředí .NET 4.0 frameworku.

Klíčová slova: bin, packing, optimalizace nákladu, knapsack, container loading problem, stromová struktura, quartz .NET, EPPlus, přeprava

Abstract

This thesis aims on problem of truck loading in it most efficient form, which means fast and accurate. We are solving problem of saving the largest possible number of objects (boxes) on the minimum number of pallets, which are then transported. Once it was necessary to rely on the good estimate, but today we can solve this issue because of computers more efficiently. Modern time demands speed, efficiency and low cost. Both large and small transport companies aims to achieve that. As a result of this problem, computer industry dedicated years of research to this sector and try to come up with better methods to solve the problem. Basis of our solution is based on previous studies of authors David Pisinger, Silvano Martello and Daniele Vigo [4] from 1997, which described problem called Binpacking, which is one of methods of optimization of cargo. This bachelor work is written using LaTeXand implemented in programming language C# using .NET framework version 4.0.

Keywords: Bin, Packing, Cargo Optimization, Knapsack, Container Loading Problem, Tree Structure, Quartz .NET, EPPlus, Transport

Seznam použitých zkratek a symbolů

1D	–	One dimensional
2D	–	Two dimensional
3D	–	Three dimensional
MS	–	Microsoft
UI	–	User Interface
HTML	–	HyperText Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Už od nepaměti lidstvo trápí problém jak naložit co nejvíce na co nejméně prostoru. Kdysi bylo nutné spolehnout se jen na dobrý odhad, ale dnes můžeme tuto problematiku díky počítačům řešit mnohem efektivněji. Moderní doba vyžaduje rychlost, efektivitu a nízké ceny. Toho se snaží dosáhnout i všechny malé a velké firmy, které mají spojitost s transportem a expedicí. Jednou z nich je mezinárodní firma Gates Corporation s českou pobočkou Gates Hydraulics s.r.o. ve městě Karviná, zabývající se výrobou kovových součástek pro hydraulické hadice a montážní celky pro evropský trh v oblasti stavební a zemědělské techniky [1]. Pro tuto firmu vznikla následující bakalářská práce zabývající se optimalizací nákladu do kamionů pod pracovním názvem PackMan (Packing Manager). Počítačová věda se již několik let tomuto odvětví také věnuje a snaží se přijít se stále lepšími metodami, jak problém vyřešit. Existuje několik možných přístupů k problému, většina z nich má základ v heuristickém přístupu. My jsme si jako základ řešení našeho problému vybrali předešlé studie Davida Pisingra, Silvano Martello, Daniele Vigo [4] z roku 1997, kde popsali problém zvaný Bin Packing, což je jedna z metod řešení optimalizace nákladu. V této práci se zabýváme hlavně jeho 2D a 3D verzí, které v naší aplikaci pro firmu Gates Hydraulics s.r.o využijeme. V podkapitole 1.1 této práce si představíme, co znamená nedeterministicky polynomiální problém a jak jej řešit. Postupy pro řešení v čele s Bin Packing algoritmem si popíšeme v kapitole 2. V podkapitole 2.4 se zastavíme u stromových struktur. V kapitole 3 se budeme zabývat nejen implementací a optimalizací algoritmu pro řešení nejlepšího možného uložení nákladu, ale také komplexním návrhem a implementací celého webového rozhraní pro vizualizaci výsledků algoritmu, denní přehledy objednávek, nastavení celého systému ukládání a případnou zpětnou komunikaci. V podkapitole 4.6 se podíváme na několik syntetických testů, abychom porovnali různé metody řešení a zároveň se podívali na vliv nastavení jednotlivých prvků algoritmu na odlišné výsledky. V poslední kapitole 5 se lehce podíváme na možnosti řešení vizualizace daného problému pomocí aktuálních grafických frameworků jakožto Unity Engine od Unity Technologies nebo XNA Framework firmy Microsoft. Celá implementace je psána v programovacím jazyce C# v prostředí .NET 4.0 frameworku.

1.1 Náročnost problému

Řešení optimalizace nákladu na kamiony je v rámci počítačové vědy netriviální úloha. Základem pro náklad je set objektů ve tvaru obdelníků (beden), které musí být naložený na co nejmenší počet palet a ty jsou následně exportovány. Výsledkem implementovaného řešení je určení finálního množství palet, které mají být v určitý den odvezeny. Bedny, stejně jako palety jsou určeny třemi parametry. Jedná se o šířku, délku a výšku. Parametry pro každou bednu a paletu se mohou lišit. V rámci námi implementovaného řešení není možné podle požadavku s objekty rotovat. Celá problematika ukládání objektů v prostoru spadá do oblasti takzvaných kombinatorických NP-hard (těžkých) problémů [2], neboli nedeterministický polynomiální problém. Co to znamená nedeterministický polynomiální problém a jak jej řešit? Dosud neexistuje cesta, jak zcela vyřešit tento problém v rozumném čase. Jedinou možností jak řešit tento problém je tedy heuris-

tický přístup, nebo hrubá síla algoritmu a vyzkoušení všech možných kombinací uložení v naději, že se nám podaří najít co možná nejlepší rozložení nákladu. I kdybychom k výpočtům tohoto problému využili superpočítače, stále nemáme zaručeno, že všechny objekty budou na požadované palety perfektně naskládány. Na zcela přesný výpočet si tak pravděpodobně ještě několik let počkáme.

2 Možnosti algoritmů

Ještě před započetím práce na algoritmu a řešením celého systému pro optimalizaci bylo potřeba provést průzkum, na jehož základě jsme se poté rozhodli využít určitý algoritmus, kterým, jak již bylo řečeno výše, je Bin Packing. Algoritmů, které určitým způsobem řeší ukládání objektů je mnoho, nicméně zdaleka všechny se nezabývají přímo ukládáním na palety. Museli jsme také vzít v úvahu požadavky firmy. Mezi hlavní požadavky patří ukládání beden na několik různě velkých palet nebo nemožnost otáčení beden. V následujícím textu si popíšeme jednotlivé možnosti řešení ukládání objektů a zdůvodníme, proč se hodí či nehodí pro naše požadavky.

2.1 Knapsack problem

Knapsack [3], nebo také Rucksack problém je jedním z nejzákladnějších problémů kombinatorické optimalizace. Představme si, že máme určitý počet objektů. Každý z těchto objektů má svou váhu a hodnotu. Algoritmus se snaží určit počet jednotlivých položek v kolekci tak, aby zůstala jen taková množina objektů, které mají co nejnižší, nebo maximálně stejnou váhu jako je daný celkový limit váhy a zároveň mají co největší hodnotu. Jméno algoritmu vychází z problému naplnění batohu s pevně danými rozměry těmi nejdůležitějšími věcmi. Knapsack algoritmus je uveden pouze pro ukázku. V této bakalářské práci se dále nevyskytuje. Algoritmus se nám po přezkoumání jevil jako nevhodný. Řeší sice uspořádání předmětů v prostoru, ale pouze na základě váhy a hodnoty objektu, a proto jsem od něj upustili. Knapsack je dlouze zkoumaným problémem. Velmi dobře je popsán v práci pana Davida Pisingera zde [6].

2.2 Container loading problem

Tato verze algoritmu se od Knapsack problému výrazně liší. V provedení tohoto algoritmu musí být všechny objekty (bedny) naloženy do jediného kontejneru (např. lodní kontejner). Tento kontejner může mít teoreticky nekonečnou délku, nicméně, vždy se musíme snažit najít realizovatelné řešení, ve kterém budou naloženy všechny předměty a zároveň délka kontejneru bude co možná nejkratší. Velmi dobře je tento typ problému popsán v práci autorů E.E. Bischoff a M.D. Marriott [8]. Mimo jiné je v této práci prezentováno 14 různých heuristických metod založených na výzkumu J.A. George a D.F. Robinson [9]. Container loading problém ve své podstatě řeší problém nákladu do kamionu, nicméně velmi omezeným způsobem díky možnosti ukládat pouze do jednoho teoreticky nekonečně dlouhého kontejneru. Tento problém by se vhodným programátorským postupem dal jistě obejít, nicméně v dnešní době již existují jiné algoritmy, které jsou pro naše využití ve firmě Gates Hydraulics s.r.o. vhodnější a proto jsme se rozhodli tento algoritmus také nevyužít. V následující kapitole se již podíváme na Bin Packing problem.

2.3 Bin Packing Problem

Bin Packing problém [4]je problém známý svou obtížností jako NP-těžký (nedeterministicky-polynomiální problém), jehož základním cílem je naskládat objekty různých velikostí do konečného počtu kontejnerů tak, aby bylo použito co nejmenší množství těchto kontejnerů v co nejkratším čase a s minimálním množstvím nepoužité plochy. I přes svou obtížnost se tento typ algoritmu hojně vyskytuje v praxi. Jde především o lodní přístavy (plnění kontejnerů), nakládání palet do kamionů, nebo například mechanický a elektronický design. Rozdíl mezi Bin Packing a Container Loading problémem je v tom, že Bin Packing dovoluje přímo počítat s více možnými kontejnery, kdežto Container Loading pouze s jedním. Bin packing problém má několik variant, které si na následujících řádcích popíšeme.

2.3.1 One Dimensional Bin Packing Problem

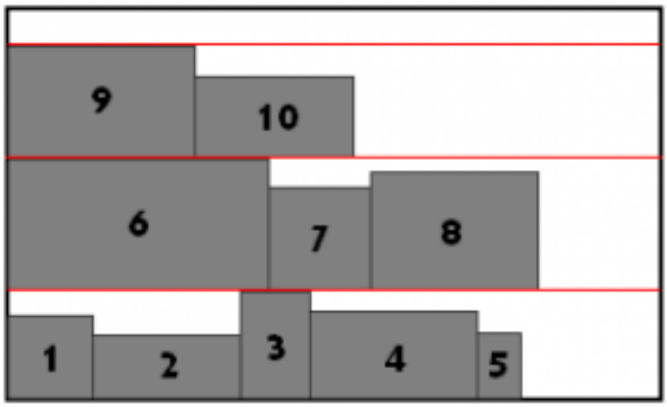
Jedná se o nejjednodušší formu tohoto problému. Příklad 1D Bin Packing problému si můžeme představit jako televizní reklamy při komerční přestávce. Existuje určité množství natočených reklamních spotů různých délek, které musí být přiřazeny do komerční přestávky. Komerční přestávka nesmí trvat déle, než 5 minut. Toto je typický problém, který se dá řešit pomocí One-Dimensional Bin Packingu.

2.3.2 Two Dimensional Bin Packing Problem

V druhé variantě tohoto problému se jedná v nejvíce případech o naskládání různě velkých objektů (většinou obdelníků) specifikovanými svou šířkou a délkou do nejmenšího možného množství kontejnerů pevně daných proporcí. Žádný z objektů se nesmí protínat a nesmí být uvnitř dalšího objektu. Příkladem může být již zmíněné skládání palet do kamionů nebo například herní průmysl, kde se tímto způsobem z několika stovek miniaturních textur dělají velké textury, které výrazně šetří výkon počítače. Tato varianta problému je popsána několika způsoby řešení, z nichž si představíme pouze několik vybraných, protože tato verze problému není hlavním námětem bakalářské práce.

2.3.2.1 Shelf Algoritmus Tento algoritmus je jeden z nejjednodušších [10], který se pro tuto formu problému dá použít. Varianty jsou asymptotické algoritmy. V rámci tohoto algoritmu rozdělíme kontejner horizontálně na několik polic a na tyto police poté pokládáme objekty. Police má atributy šířky a délky. Když dojde k tomu, že se první police zaplní, vytvoříme nad ní novou polici a začneme skládat objekty tam. Pokud nastane případ, že by vkládaný objekt mohl mít parametry (délka, šířka) na vložení do více polic, využijeme v tomto případě heuristického přístupu a rozhodneme, do které police objekt vložíme.

Shelf algoritmus nedosahuje nejlepších výsledků optimalizace [10]. Díky tomu, že vyplňuje objekty plochu pouze v rámci polic, zanechává mezi objekty spoustu volného místa. Lépe problém vysvětlí obrázek 1. Mimoto je algoritmus velmi citlivý na pořadí



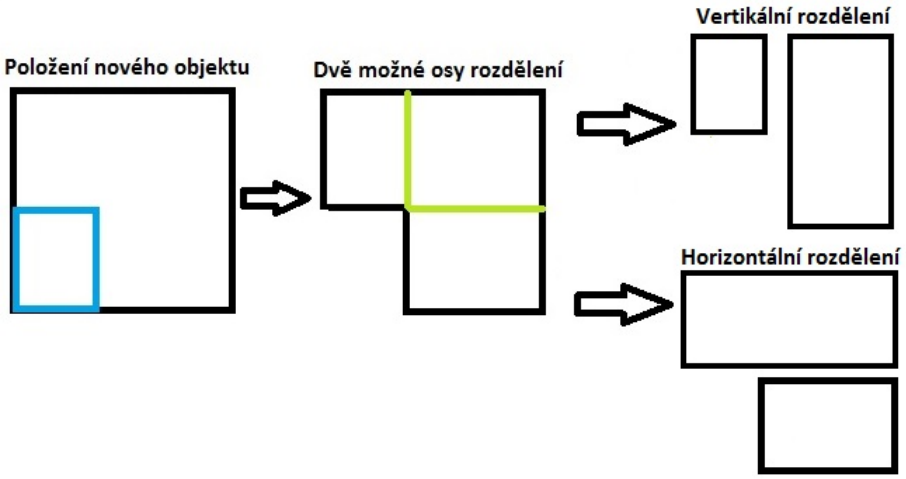
Obrázek 1: Ukázka výpočtu pomocí Shelf algoritmu. Zdroj:[10]

vložených objektů, i když jsou vložené předměty velikostně téměř stejné. Ve většině případu lze využít lepších řadících metod.

2.3.2.2 Guillotine Algoritmus V tomto případě je zvolen zcela jiný přístup k problému. Guillotine algoritmus [10] je založen na operaci, která se nazývá guillotine split placement, což je v principu vložení objektu do rohu volného kontejneru. Poté je zbývající plocha ve tvaru písmene „L“ opět rozdělena na dvě prázdné plochy kontejneru, kde se dále vkládají další objekty, a poté se opět rozdělí zbývající plocha. Podle velikosti objektu je poté vždy zvolena plocha, do které je umístěn. Algoritmy založené na principu rozdělení zbývající plochy jsou velmi známé a hodně používané. Příkladem může být již zmíněné texturování v herním průmyslu, kde se používají algoritmy s tímto principem. Na obrázku 2 pod tímto textem jsou vidět červené linky, jakožto znázornění rozdělených ploch poté, co algoritmus vloží objekty.

Obrázek 2: Ukázka výpočtu pomocí Guillotine algoritmu. Zdroj:[10]

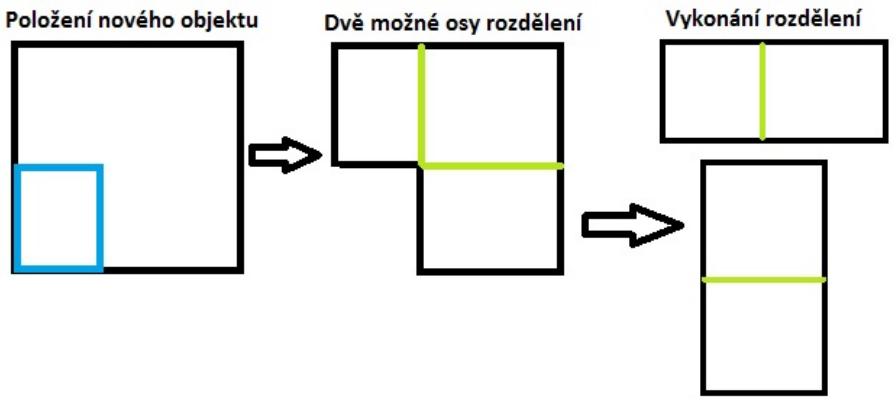
Výhodou Guillotine algoritmu je, že si uchovává přesné informace o volném místě zbývající plochy, což je další velký rozdíl oproti Shelf algoritmu. Nevýhodou algoritmu je, že bere v úvahu pouze umístění, ve kterém je možno objekt pouze plně obsadit do některé volné plochy. Nikdy se nestane případ, že by byl objekt vložen do místa, kde by mohl přetnout dvě volné plochy.



Obrázek 3: Proces rozdělení plochy po vložení prvního objektu při využití Guillotine alg.

2.3.2.3 Maximal Rectangle Algoritmus Třetím algoritmem, který řeší 2D Bin Packing je Maximal Rectangle algoritmus [10]. Tento algoritmus má hodně společného s předešlým Guillotine algoritmem a navíc přidává další způsob řešení problému. Stejně jako Guillotine algoritmus, Maximal Rectangles algoritmus si ukládá list volných ploch, které reprezentují zbývající volnou oblast kontejneru. Změna oproti Guillotine algoritmu přichází ve výběru volné plochy k uložení dalšího objektu. Na rozdíl od Guillotine algoritmu, který vybírá jednu ze dvou dělených os, na kterou poté ukládá další objekt, Maximal Rectangle vykoná operaci, která v podstatě odpovídá na výběr obou dělených os současně. Více napoví obrázek 4 pod textem.

Když vložíme objekt do levého dolního rohu prázdného kontejneru, vypočítáme dvě volné plochy, které pokrývají volnou oblast ve tvaru písmene „L“. Tyto dvě volné plochy mají maximální délku v každém směru. Což znamená, že na každé straně se dotýkají buď hrany kontejneru, nebo nějakého již předtím vloženého objektu. To nám zaručí, že když se porovnávají potencionální pozice k vložení objektu, můžeme zvážit každou volnou plochu postupně a tím si budeme jistí, že pokud se podaří objekt uložit, nepřeskočíme žádnou správnou pozici pro uložení. Tento algoritmus má i nevýhodu. Objekty jsou po dvou disjunktní a tím vytváříme problém při vkládání dalších objektů. Tento problém vzniká poté, co vložíme objekt do kontejneru a musíme zkontrolovat a aktualizovat všechny ostatní objekty, pro které platí, že průnik objektů dohromady s kontejnerem se nerovná 0, jinak se data stanou nekonzistentní. Na řešení této situace využijeme způ-



Obrázek 4: Proces rozdělení plochy po vložení prvního objektu při využití MaxRect alg.

sob, ve kterém se jednoduše posouváme po každé volné ploše v kontejneru a protínáme ji s objekty. Tím vytvoříme sadu nových volných ploch v kontejneru. Po tomto kroku nám mohou vzniknout degenerované nebo nemaximální volné plochy, proto opět projdeme všechny volné plochy, které jsou uvnitř kontejneru a smažeme je, pokud zde zároveň existuje další volná plocha. Bohužel, díky tomu, že algoritmus několikrát prochází všechny prvky v kontejneru, jeho časová náročnost se zvyšuje.

2.3.2.4 Skyline Algoritmus Díky tomu, že Maximal Rectangles algoritmus spoustu času tráví manipulací se seznamem, který musí udržovat, vznikl Skyline algoritmus, který zjednodušuje celý proces a zároveň u něj lze taktéž provádět heuristické výpočty od levého-dolního rohu. Skyline algoritmus je na druhou stranu stejně ztrátový jako Shelf algoritmus, protože si „nepamatuje“ volné oblasti kontejneru a některé volné plochy může označit jako použité pro objekty. Jako kompromis mezi těmito dvěma vlastnostmi funguje Skyline algoritmus výrazně rychleji než algoritmy využívající Maximal Rectangles. Skyline algoritmus funguje na principu udržování listu „horizontů“. Znamená to, že si ukládá pouze pozice vytvořené nejvyššími hranami již vložených objektů. Takový list je velmi jednoduché spravovat a roste lineárně s počtem objektů, které bylo do kontejneru vloženy.

Zde jsme si představili 4 základní postupy, jak řešit dvoudimenzionální Bin Packing problém. Algoritmy jsou popsány obecně a všechny 4 postupy mají své další modifikace a úpravy, které vedou k lepším/jiným výsledkům. Jako nejvhodnější algoritmy pro řešení dvou-rozměrného Bin Packing problému se jeví Guillotine a Maximal Rectangles algoritmy. Shelf a Skyline algoritmy podávají horší výsledky a díky tomu, že si „nepamatují“ volné plochy, jejich použití není vhodné bez dalších úprav. Více o tomto tématu pojednává velmi obsáhlá studie Jukka Jylanki z roku 2010 [10]. V této studii jsou rozepsány možnosti všech výše zmíněných algoritmů, včetně jejich mírných optimalizací, úprav a také srovnání.

2.3.3 Three Dimensional Bin Packing Problem

Dále se budeme zabývat poslední variantou bin packing problému, o které je převážná část této bakalářské práce a to právě Three-Dimensional Bin Packing problém. Ze všech tří variant je právě tato varianta problému nejvíce komplexní. Díky další přidané ose se zvyšuje jak výpočetní tak časová náročnost na řešení problému. V praxi je problém velmi těžké řešit, při programování troj-rozměrného systému se používají taktéž heuristické přístupy stejně jako v předchozích případech. Obyčejně se potýkáme s přesnosti kolem 85 – 95 % oproti reálným hodnotám optimálního uložení. 3D varianta Bin Packing problému dědí spoustu vlastností od své 2D verze. Nicméně, díky přidané třetí ose vznikají další možnosti řešení. V další části textu se na pár z nich podíváme.

2.3.3.1 3D Next Fit Tento algoritmus je pravděpodobně tím nejzákladnějším pro řešení 3D Bin Packingu [7]. Algoritmus začne pracovat s první bednou a uloží ji na defaultní pozici (nejčastěji souřadnice 0, 0, 0). Poté co uloží první bednu, vezme si další bednu a uloží ji na stejnou úroveň vedle předchozí bedny. Tento postup se opakuje tak dlouho, dokud není překročena šířka kontejneru. Pokud dojde k překročení šířky, je vytvořena nová úroveň nad tou stávající a skládání beden dále pokračuje. Nejvyšší bedna z předchozí úrovně určuje základ pro nově vytvořenou úroveň. Pokud kdykoliv při skládání nastane stav, ve kterém dojde k překročení délky kontejneru, vytvoří se nová úroveň.

2.3.3.2 3D First Fit 3D First Fit [7] algoritmus je velmi podobný 3D Next Fit algoritmu. Také začíná na defaultně zvolené pozici kontejneru (souřadnice 0, 0, 0), nicméně rozdíl nastává v dalším skládání beden. NExt-Fit algoritmus položil další bednu hned vedle předchozí na stejné úrovni. 3D First Fit algoritmus se v tomto liší a místo uložení hned vedle předchozí bedny dojde k prohledání celého kontejneru od začátku na všech vytvořených úrovních, dokud není nalezeno nějaké prázdné místo, kam se vleze tato bedna. Stejně jako u předchozího algoritmu, ani zde není možné, aby šířka krabice překročila šířku celého kontejneru. Může nastat případ, kdy bedna překročí délku některé z vytvořených úrovní. V takovém případě je možné dále pokračovat za předpokladu, že součet všech délek úrovní nepřekročil stanovenou maximální délku kontejneru.

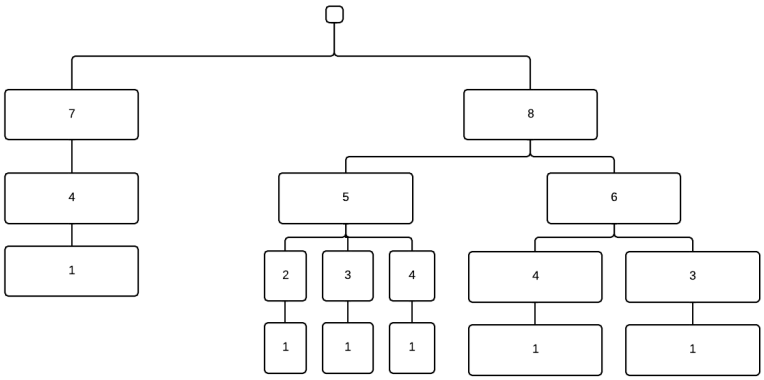
2.4 Stromová struktura

V IT existují nejrůznější datové struktury. Patří sem i stromové struktury [5], jež jsou jednou z nejvíce pokročilých a nejsilnějších nelineárních datových struktur. Nelineární strukturou rozumíme takovou organizaci prvků, která je bohatší než klasická organizace prvků v posloupnosti, kde existuje pouze relace „před“ a „za“. Ve stromové struktuře jsou všechny vztahy a prvky hierarchicky uspořádané a vznikají i relace typu „nad“ nebo „pod“.Stromové struktury jsou často využívané a aplikovatelné na celou řadu problémů a také byly použity v rámci implementace této bakalářské práce. Vztahy mezi jednotlivými položkami v této struktuře odpovídají větvím stromu. Odtud také vzniklo jméno a celou hierarchii lépe vysvětlí obrázek [**?**].

Základní struktura stromu se skládá z následujících prvků:

- kořen (root),
- vnitřní uzel (node),
- vnější uzel (leaf).

Každý strom má svůj kořen (root), který se v něm nachází vždy pouze jednou jako nejvyšší uzel ve stromu a je to také jediný prvek stromu bez rodiče. Dále obsahuje stromová struktura vnitřní uzly (nodes). Jedná se o takové uzly stromů, které ještě nejsou koncové a jsou na ně tedy navázány další uzly jako potomci. Jako poslední základní stavební kámen stromu existují Koncové uzly (leafs), což je takový prvek, který již nemá žádného dalšího potomka. V některých situacích může dojít k tomu, že má strom pouze jeden uzel. V tomto případě je pak uzel nejen kořenem stromu, ale také jeho listem.



Obrázek 5: Příklad stromové struktury

3 Vývoj

Hlavní požadavkem firmy Gates Hydraulics s.r.o. bylo vymyslet systém, který bude schopen na aktuální den vypočítat veškeré objednávky a díky tomu říci skladníkům, kolik přesně palet bude v daný den odjíždět. Podle těchto výpočtu se dále bude objednávat doprava na další dny. Výsledkem a dlouhodobým měřítkem úspěšnosti našeho řešení je snížení nákladů na dopravu pro celou firmu. Rozsahem celé práce je tedy plně automatický systém (pojmenovaný PACKMAN – zkratka pro Packing Manager) fungující v intranetu firmy, založený zejména na technologii společnosti Microsoft, čímž je .NET 4.0 framework a implementace je psána v jazyce C#.

3.1 Agilní vývoj

Od prvotní analýzy s firmou Gates s.r.o. jsme si určili, jakým způsobem bude probíhat vývoj aplikace - agilně. Software lze samozřejmě vyvíjet celou řadou způsobů, od klasického Waterfall modelu [16] (analýza, návrh, alphaverze, betaverze, RC verze, finální produkt) až po agilní metodiku. Vždy je ale nutné soustředit se na to, aby byl brán v potaz zájem zákazníka a také byl kladen důraz na kvalitu a samotný produkt. Agilní vývoj byl zvolen z důvodu nejasnosti všech zákoutí zadání z počáteční fáze vývoje. Časté změny nebo nová funkcionalita tak pro nás nebyla žádný problém. Výhodou agilního vývoje je právě jeho vývojový cyklus, který se skládá ze 4 základních kroků. Nejprve jde o tzv. nultou iteraci, což je krátká analýza a následně se naprogramuje nějaká základní část aplikace. Jde o to, aby na konci této fáze existoval nějaký počáteční produkt, od kterého se bude vše dále odvíjet a je možné jej předvést zákazníkovi. Po nulté iteraci přichází na řadu druhý krok, kterým je analýza změn. V tomto kroku se zabýváme analýzou změn, případně novou funkcionalitou a vybíráme, co dostane přednost v implementaci. Třetím krokem je pak samotná implementace zvolených prvků a jako poslední část celého kola je představení produktu zákazníkovi. Pokud při čtvrtém kroku nejsme na konci, ale přidáváme nebo upravujeme další funkcionalitu, vracíme se ke kroku dva a opět opakujeme. Pokud je čtvrtý krok finální, vývoj končí. Tyto čtyři body se dají rozšířit ještě o jeden poslední, pátý. Zde se jedná především o post údržbu a případný budoucí rozvoj aplikace, který opět běží v těchto cyklech. První čtyři body se označují jako jedna iterace a opakujeme je tak dlouho, dokud není náš cíl (aplikace) ukončen. Agilní metodika vývoje má několik výhod. Mezi největší patří bezproblémové změny zadání za běhu, dále pak zrychlení doručování produktu a častější kontakt zákazníka s produktem, z čehož následně také vyplývá častější komunikace se zákazníkem a tedy jeho celková spokojenost.

3.1.1 Fáze jednotlivých iterací

3.1.1.1 Analýzy změny V této fázi dostává tým k dispozici klientovi další priority pro následující cyklus. Někdy se dopředu ví, na čem se bude pracovat dalších několik iterací, nebo nejsou dokončené práce z poslední iterace. Zohledňují se i úpravy pro lepší kompatibilitu s novými změnami. Dopředu je stanoveno, které práce by měli být provedeny.

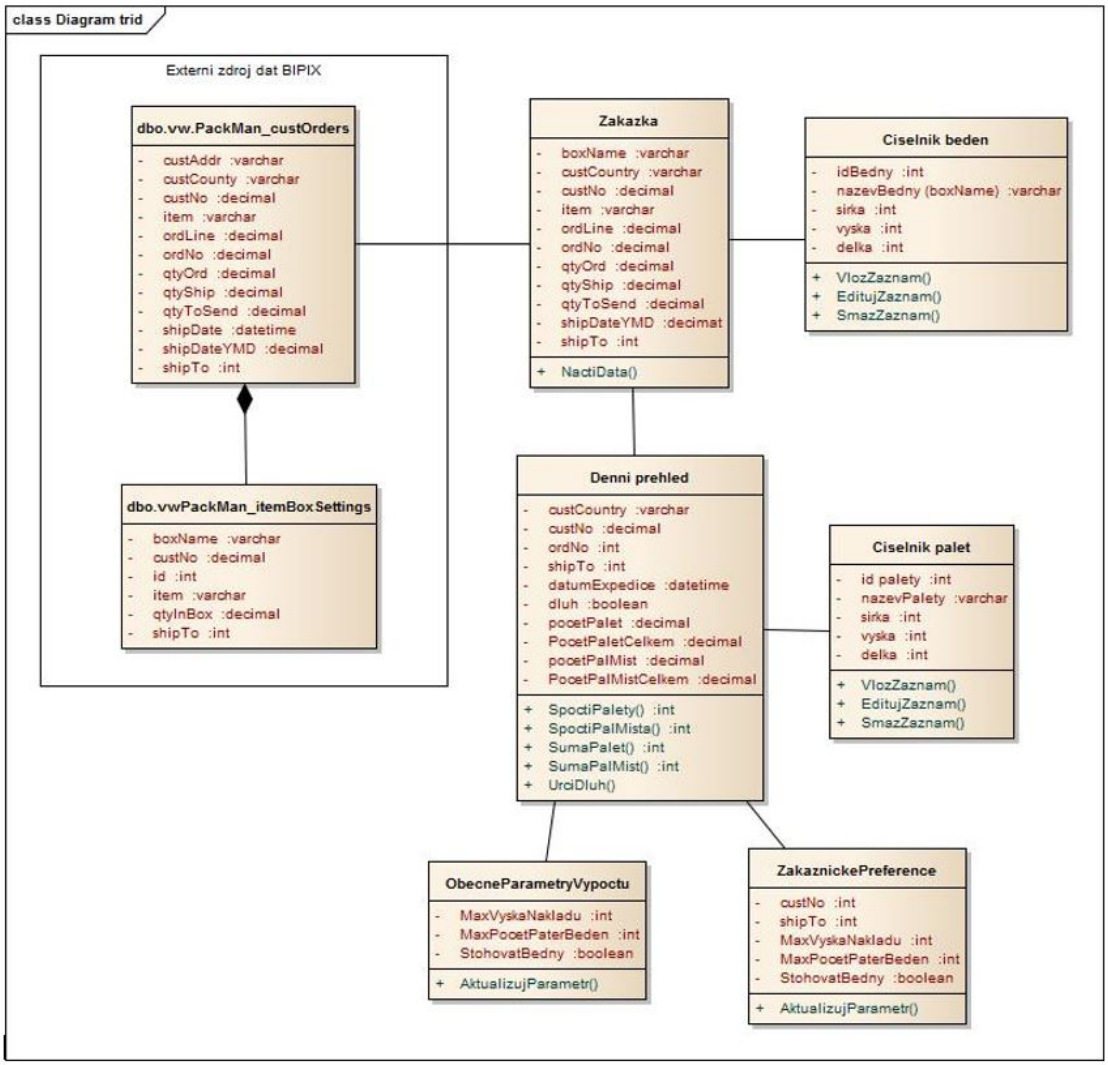
Také se často v této fázi používá tzv. prototypování, aby se ujasnilo, jak bude vypadat výsledek. Pokud je to nutné, model celého řešení (aplikace, webu, produktu) je upraven tak, aby vyhovoval změnám. V případě klasického agilního vývoje se v tomto kroku dělají také akceptační scénáře a testy, změny datových modelů, úpravy a přidávání logiky, případně úpravy pomocných knihoven a spoustu dalších.

3.1.1.2 Implementace změny V této části se jedná především o dění v týmu. Je nutné pravidelně kontrolovat výstupy, plnění daných úkolů, komunikovat další změny a zároveň také odstraňovat problémy, které mohou bránit týmu v práci. Agilní přístup nehraje na klasickou korporátní strukturu vedení týmů, nýbrž na rovnocenné postavení v týmu, kde všichni společně pracují na jedné věci a tím je dodržení obsahu na čas. Týmy jsou většinou velmi dobře sebeřízené.

3.1.1.3 Předvedení nových změn Pravidlem agilního vývoje bývá, že zákazníkovi je ukázána pouze skutečně dokončená práce. Všechny nehotové části nebo vlastnosti jsou skryty z grafického rozhraní pryč. Důležité pro zákazníka je vidět viditelné změny. Vždy se nemusí jednat pouze o změny grafického typu, mimo jiné mlže jít například o zrychlení aplikace, lepší práci s databází, ušetření zdrojů, či lepší optimalizace. To, že jsou nedokončené věci skryty z grafického prostředí, neznamená, že zákazník neví o všech nedodělcích. Právě naopak, klient by měl vědět o všech aktuálních nedodělcích či rozdělané práci mimo ukázku. V této fázi zákazník hodnotí změny za daný cyklus a vybírá, jaká implementace bude následovat.

3.2 Databázová struktura

Pro potřeby fungování algoritmu bylo nutné vyřešit také databázovou strukturu, tak aby vyhovovala všem požadavkům. Firma Gates s.r.o. samozřejmě využívá svůj vlastní databázový server, na kterém má umístěné všechny data. Na nás bylo vymyslet strukturu pro nově vytvořený program. Dřívější struktura samozřejmě nepočítala s novým systémem na optimalizaci nákladu, proto vznikly dvě nové databázové tabulky, které má firma plně pod kontrolou a jedna upravená stávající tabulka včetně pohledu. Nově vytvořené tabulky obsahují data o rozměrech beden a palet. Tabulky jsou rozděleny, aby nedocházelo k matení a případnému narušení fungování programu, a také díky požadavku na změny hodnot v tabulkách z klientské strany. Díky tomu je vše na první pohled jasné a nemělo by dojít k žádné případné ztrátě dat. V budoucnu je navíc operace přidání nových beden či palet naprosto bez problému, stejně tak jako vymazání existujících záznamů. Nově vytvořené tabulky obsahují hodnoty jako ID, jméno, rozměry, stohovatelnost. Přesný popis a názvy jednotlivých sloupců vidíte níže na obrázku.



Obrázek 6: Diagram znázorňující kompletní řešení projektu PackMan.

3.3 Specifické technologie

Při práci na tomto projektu jsme se několikrát dostali do bodu, kdy jsme obdrželi určité požadavky, u kterých jsme zprvu nevěděli jak vyřešit jejich implementaci. Většinou z důvodu nevhodné nebo nedostatečné základní funkčnosti ze strany technologie .NET frameworku. Proto jsme sáhli po knihovnách třetích stran, které nám dovolili tyto požadavky přeměnit z požadavku na skutečnost zabudovanou v systému. Všechny knihovny jsou Open-Source, jsou tedy volně šířitelné a upravitelné. Nejsme vázání žádnou licencí a můžeme je svobodně využívat. V tomto projektu jsme využili dvou malých doprovodných .NET knihoven, a jednoho mini jQuery skriptu, jedná se o:

- Quartz Scheduler Enterprise .NET,
- EPPlus,
- jQuery Block UI Plugin.

3.3.1 Quartz Scheduler

Quartz Enterprise Scheduler .NET (dále Quartz.NET) je open-source .NET knihovna napsaná v jazyce C# vycházející z její původní velmi populární verze v jazyce Java napsané vývojářem Jamesem Housem. [15] Tato knihovna rozšiřuje funkcionalitu frameworku .NET o lepší plánování úloh v kódu. Umožňuje plánování úloh, spouštění úloh a také jejich perzistenci. Quartz.NET úlohy se spouští na základě výskytu triggeru. Triggery můžou být vytvořený v podstatě libovolnou kombinací těchto směrnic:

- Určitá denní doba (na milisekundy),
- určitý den v týdnu,
- určitý den v měsíci,
- určitý den v roce,
- opakování určitého počtu opakování,
- opakování do specifického času/data,
- opakování do nekonečna,
- opakování s časovou prodlevou.

Tuto knihovnu jsem ve své bakalářské práci využil pro funkci plánování času automatického exportu souborů na server. Požadavkem firmy bylo automatické každodenní zálohování dat z databáze ve formě Excel souboru na server a možnost nastavení času, kdy se bude soubor generovat. V základním nastavení aplikace je tato možnost a uživatel tak může nastavit libovolný čas, kdy se bude spouštět trigger a zálohovat na server. Knihovna Quartz.NET v tomto případě velmi pomohla s implementací a řešením problému. Stačilo pouze napsat metodu, která reaguje na proměnnou zadanou od uživatele a podle toho se spouští.

3.3.2 EPPlus

EPPlus je .NET knihovna umožňující čtení a zápis do Excel 2007/2010 souborů využívající Open Office Xml formát (xlxs). Tato knihovna je napsána pro pokročilejší práci s Excel soubory na serveru. EPPlus podporuje rozsahy buněk, stylování buněk (okraje, barvy, výplně, pásma, čísla, zarovnání), grafy, obrázky, tvary, komentáře, tabulky, ochranu souboru, šifrování, kontingenční tabulky, ověření dat, podmíněné formátování a mnoho dalšího. V jednotkách sekund je EPPlus schopná zpracovat až 50 000 buněk. Knihovna má

kompletní .NET integraci. Tato knihovna mi taktéž pomohla v řešení exportů Excel souborů na server. Konkrétně jsem si v tomto případě určil jak se bude soubor exportovat, jaké bude mít vygenerované jméno a také jsem kompletně k obrazu firmy Gates Hydraulics s.r.o. upravil formát samotného Excel souboru, tak aby byl přehledný a pasoval k firemní kultuře. S knihovnou se velmi dobře pracuje, je flexibilní a poskytuje opravdu velké množství možností úprav, které standardní .NET framework nemá.

```
FileInfo  newFile = new FileInfo(outputDir.FullName + @"\sample.xlsx");

          ExcelPackage pck = new ExcelPackage(newFile);

          var  ws = pck.Workbook.Worksheets.Add("Content");
          ws.View.ShowGridLines = false;
```

Výpis 1: Příklad vytvoření Excel sešitu pomocí EPPlus

V přiložené ukázce kódu je vidět tvorba nového excel souboru pomocí knihovny EPPlus. Nejprve je nutné si vytvořit nový soubor pomocí nového objektu typu FileInfo se jménem „sample.xlsx“. Poté vytvoříme instanci objektu ExcelPackage a můžeme přímo začít pracovat na formátování a naplnění buněk do Excelu. V této ukázce by například funkce ShowGridLines = false zajistili, že se mezi buňkami nově vytvořeného excel souboru nebude zobrazovat ohraničení.

3.3.3 jQuery Block UI Plugin

Tento malý skript simuluje synchronní chování při použití technologie AJAX, bez toho, aniž by uzamknul prohlížeč. K uzamknutí prohlížeče standardně dochází v momentě, kdy vyšleme XMLHttpRequest objekt, který v synchronním módu zapříčiní „zamrznutí“ celého prohlížeče do doby, než je tento vzdálený požadavek proveden. Takové chování většinou není žádoucí, a proto přišel na řadu tento malý šikovný plugin. Jakmile je BlockUI plugin aktivní, bude bránit činnosti uživatele se stránkou (nebo její částí), dokud opět není deaktivován. V našem případě byl plugin využit na všechny tabulky, které jsou editovatelné z klientské strany. Díky tomuto pluginu není třeba při změně parametrů v tabulce znovu aktualizovat celou stránku, ale aktualizuje se pouze element zobrazující tabulku, v našem případě GridView.

Příklad 3.1
Uveďme si příklad zablokování určitého elementu ve webové prezentaci. Ve zdrojovém kódu níže je popsáno, jak lze jednoduchým způsobem zablokovat celý element, včetně všech objektů v něm se vyskytujících. Pro demonstraci fungování pluginu jsou tu tři tlačítka. Block, Block with message a Unblock. Po kliknutí na tlačítko block, se celý element zablokuje a stylově začerní. Pokud klikneme na tlačítko Block with message, nejen že se element zablokuje a opět začerní, ale také nepřetržitě ukazuje námi zvolenou zprávu (v tomto případě Proccessing). Funkce tlačítka Unblock je pak již víceméně jasná, dojde k opětovnému odblokování celého elementu.

```
<script type="text/javascript">
 $(document).ready(function() {

      $('#blockButton').click(function() {
          $('div.test').block({ message: null });
      });

      $('#blockButton2').click(function() {
          $('div.test').block({
              message: '<h1>Processing</h1>',
              css: { border: '3px solid #a00' }
          });
      });

      $('#unblockButton').click(function() {
          $('div.test').unblock();
      });

      $('a.test').click(function() {
          alert('link clicked');
          return false;
      });
  });
</script>
```

Výpis 2: Příklad blokování elementu pluginem jQuery BlockUI

4 Řešení optimalizace nákladu

Pro řešení hlavního problému - jak počítat optimalizaci nákladu - jsme provedli dřívější analýzu. Nejprve jsme se dostali k algoritmu nazývanému Knapsack problem. Ze začátku vypadal tento algoritmus jako to pravé pro naše účely, ale s rozšiřujícími se požadavky a ujasněním, že budeme počítat náklad ve třech dimenzích jsme od tohoto řešení nakonec ustoupili a podívali se po dalších algoritmech. Narazili jsem na algoritmus pojmenovaný jako Container Loading problem, který se blížil našim požadavkům mnohem více. Jeho limitace na jeden kontejner nás bohužel opět uvrhla do hledání další možnosti. Nakonec jsme se dostali k Bin Packing algoritmu, který používáme jako aktuální řešení. Tato volba se jeví jako velmi dobrá, protože v případě potřeby pokrývá jak 2D, tak 3D variantu, které v práci využíváme. Projekt Packman obsahuje 4 hlavní webové stránky – denní přehled, archív, nastavení systému a kontaktní formulář.

4.1 Algoritmus

Celý proces výpočtu začíná v databázi firmy Gates Hydraulics s.r.o. Poté co uživatel otevře webové rozhraní programu, jsou do hlavní tabulky pro aktuální den načteny všechny data pomocí technologie Linq to SQL, týkající se konkrétního dne a ve stejný okamžik se ty samé data ukládají do datové struktury programu. Pro algoritmus je důležitých několik prvků, jedná se hlavně o typ krabice a její rozměry, případně stohovatelnost. Poté co jsou data uloženy, program si data převezme a využije je pro následující manupulaci. Nejprve je nutné vypočítat si množství krabic, které budou expedovány. Každý produkt má určenou svou krabici a počet kusů, který do ní vleze. Jednoduchými matematickými operacemi se tedy dostaneme ke konečnému počtu konkrétních typů beden. Jakmile víme, jaké typy krabic a kolik kusů v aktuálním dni expedujeme, můžeme začít počítat optimální naložení nákladu. Je taktéž nutné načíst si parametry palet, na které budou bedny ukládány. Každá bedna má svou šířku $width$, délku $height$ a výšku $depth$. Stejné parametry mají také palety.

V tomto momentě ví algoritmus o celkovém počtu krabic, které budou expedovány a jejich rozměry. První bedna je vždy uložena na souřadnice 0, 0, 0 (x, y, z osy). Nejprve v první metodě dle načtených parametrů vytvoří první bednu s parametry width, height, depth a porovná ji s parametry palety. Po té co je bedna vložena do palety, uloží se do proměnných všechny hodnoty týkající se umístění aktuální bedny. Algoritmus začne načítat jednu bednu podruhé. Poté co načte další bednu a její rozměry, podívá se na uložení předešlé bedny a aktuální bednu přiřadí na osu x vedle předešlé bedny. Neustále je nutné porovnávat, zda krabice již nepřekročila limit stanovené palety. Krabice jsou postupně skládány po ose x (width), až do doby, než v tomto směru nelze další krabici přidat. V takovém případě se další krabice uloží ve směru osy y, souřadnice osy x je vynulována a pokračuje stejný postup. Program pozná, že došlo k překročení hranice palety díky proměnné CurrentX (Y, Z), které v sobě sčítají hodnoty všech krabic v určitém směru. Když dojde na stav, že již nelze přidat krabici ani ve směru osy x ani ve směru osy y, dojde k uložení krabice na předešlou krabici po ose Z a ostatní dvě osy jsou opět vynulovány, aby bylo možné přidávat další krabice po jejích osách, ale již ve zvýšené úrovni na pa-

letě. Celý algoritmus funguje ve stromové struktuře, kde má každý uzel možnost až tří potomků. Levý, pravý a horní. Díky tomu si stále pamatuje, kde je jaká krabice a její parametry a dá se neustále rekurzivně procházet. Tímto způsobem algoritmus pokračuje až do doby, než jsou vyskládány všechny krabice a je možno sdělit výsledek.

4.2 Denní přehled

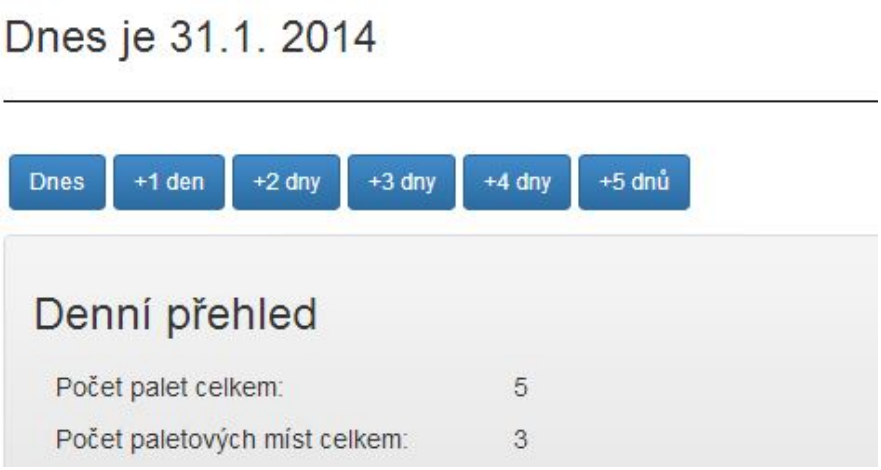
První ze čtyř podstránek celého PackMan projektu. Je to také hlavní část webové prezentace pro optimalizaci nákladu pro firmu Gates Hydraulics s.r.o fungující na intranetu společnosti.

Denní přehled obsahuje:

- Výpočet paletových míst na konkrétní den,
- tlačítka pro přesun až o pět dní kupředu,
- výpis aktuálních objednávek pro konkrétní den,
- výpis zpožděných objednávek ve spodní části webu.

Výpočet palet byl po sérii několika testů a domluvě naprogramován tak, aby se při každém přepnutí dne na jiné datum na pozadí opět přepočítal. Náročnost výpočtu se pohybuje ve většině případu kolem desítek beden denně, algoritmus zvládá vše spočítat a ihned zobrazit výpočet na webové stránce. V základní verzi algoritmus přepočítává všechny bedny na rozměry europalet. V budoucích požadavcích je také finální přepočet na menší palety. Doposud funguje objednávací systém firmy Gates Hydraulics s.r.o na jednoduchém principu. Doprava je objednávána tři dny dopředu, proto vznikl požadavek na přidání celkem pěti tlačítek na přesun mezi konkrétními dny (plus 1 až 5 dní). Pokud uživatel otevře aplikaci Packman v intranetu firmy, v základu vždy uvidí aktuální den. V případě, že je třeba objednat dopravu, může se díky této funkcionalitě posunout například o tři dny dopředu. Díky algoritmu má během vteřiny vypočítáno, kolik palet bude vychystáno na tento konkrétní den a je tedy ihned možné objednávat dopravu. Tyto tlačítka jsou řešeny pomocí klasických objektů „Button“, které poskytuje .NET Framework. V implementaci jednotlivých tlačítek používáme metody „DateTime“, která je pro tyto potřeby navržena. V případě tlačítek „z budoucnosti“ (+1, +2, +3, +4, +5) zde funguje funkce AddDays().

Výpis aktuálních objednávek je nejdůležitější z pohledu dat. Pro pracovníky Gates Hydraulics s.r.o. poskytuje pohled na aktuální data. Je to výpis nejdůležitějších informací týkající se aktuálních objednávek na konkrétní den. Tabulka obsahuje dle požadavků zvolené informace o zákaznících a přehled jejich aktuálních objednávek. Výpis zpožděných objednávek obsahuje – co se zákazníku týče – stejná data, nicméně hlavní funkčností je ucelený přehled zpožděných objednávek. Z důvodů chybějících součástek, chyby ve výrobě, nebo neúspěšnou prověrkou na kontrole kvality může dojít ke zpoždění objednávky. Všechny tyto objednávky jsou přehledně vypsány v této tabulce. Objednávka v této tabulce zůstane do doby, než bude kompletně vyřízena. Oba výpisy (a všechny ostatní výpisy systému) jsou řešeny pomocí technologie Microsoft SQL, mají svoji databázovou tabulku a čerpají data z interního serveru společnosti. Výpisy jsou z pohledu technologie řešeny standardním způsobem. Tabulky z MS SQL server jsou díky SQLDataAdapteru nahrány na GridView control objekty na webové stránce, kde jsou poté zobrazeny. Tyto dva výpisy nemají povolené úpravy z klientské strany, nicméně zbytek



Obrázek 7: Informace o počtu palet na aktuální den

webových tabulek takovou funkcionalitou disponuje. Zde by mohly uprávy tabulek z klientské strany napáchat více škody než užitku.

4.3 Archív

Druhá část webové prezentace Packmana je o možnostech archivace reportů. Požadavek pro tuto funkcionalitu se objevil později při vývoji. Požadavkem bylo archivování objednávek z každého dne, pokud možno automaticky. Stránka obsahuje jediný tabulkový výpis, ve kterém jsou zobrazeny reporty ve formátu Microsoft Excel, které jsou automaticky generovány každý den v dobu, kterou si zvolí sám uživatel aplikace. V základním nastavení je tato doba nastavená na 22:00 hodin každý den. Exporty do formátu Microsoft Excel jsou v korporátní sféře běžné požadavky. Framework .NET je pro tyto potřeby sice připravený, ale rozhodně ne dokonalý, proto jsem do procesu generování exportů zapojil dostupnou open source knihovnu EPPlus, vyvíjenou několika nezávislými vývojáři, která velkým způsobem rozšiřuje možnosti těchto exportů. Díky EPPlus se dá s trochou nadsázky v nastavení exportovaného souboru vyladit téměř cokoliv do posledního detailu. Více se o této knihovně rozepisuji v samotném odstavci EPPlus. [14] Technologicky je nejprve potřeba inicializovat ExcelPackage, což je nejvyšší excelovský objekt poskytující přístup ke všem částem dokumentu. Poté musíme určit, odkud budou brány data pro report. To zajišťuje metoda LoadFromDataTable(datová struktura, true). Jakmile načteme data, můžeme si díky EPPlus libovolně upravit excelovský soubor k obrazu svému. Můžeme nastavit stylování písma, buněk, barvu pozadí, barvy písma a mnoho dalšího. Pokud to situace vyžaduje, díky EPPlus jsme také mimo jiné schopni tvořit grafy přesně definované podle nás. Poslední částí pro vytvoření reportu je Response část. V této části nejprve vyčistíme veškerý obsah z Buffer Streamu díky metodě Response.Clear(), poté přidáme díky metodě ContentyType informace o tom, že se jedná o Excel souboru a přidáme hlavičku souboru v libovolně zvoleném tvaru pomocí metody AddHeader().

Nakonec vše zapíšeme díky metodě BinaryWrite() a odešleme všechen obsah uložený v BufferStreamu, což je námi vytvořený Excel soubor.

V archivování reportů poté vznikly další dva menší požadavky:

- Možnost nastavit maximální počet reportů,
- manuální nastavení času generování reportu.

První požadavek má jednoduché řešení. V aplikaci na pozadí přibyla zkouška, zda počet všech exportů dosáhl uživatelem uloženého čísla a pokud ano, nebude již tvořit nové, ale pouze dokola přepisovat určený počet exportů. Druhý požadavek má již obtížnější implementaci. Při hledání řešení jsme narazili na plugin jmenující se Quartz .NET, což je zdarma dostupná .NET knihovna starající se o plánování konkrétních úloh spouštěných na základě triggeru, tedy nějaké námi definované události. Samotný .NET framework nemá moc dobré možnosti řešení takto plánovaných úloh, proto jsme s radostí po této knihovně sáhli. V tomto případě nám Quartz .NET velmi pomohl. Jeho implementace není nijak obtížná a možnosti nastavení triggeru jsou velmi široké. Díky této knihovně se nám podařilo naimplementovat metody, které reagují na zadaný parametr uživatele a podle toho se spouští. Pro opakované spouštění se využívá metoda CronScheduleBuilder. Příklad jednoduchého triggeru, který se spustí každý den v 10:00 v podání Quartz .NET knihovny:

Příklad 4.1

```
public class Trigger()
{

    ITrigger Trigger = TriggerBuilder.Create()
            .WithIdentity("trigger3", "group")
            .WithSchedule(CronScheduleBuilder.DailyAtHourAndMinute(10, 00))
            .ForJob(myJobKey)
            .Build();

}
```

Výpis 3: Příklad CronScheduler jobu

■

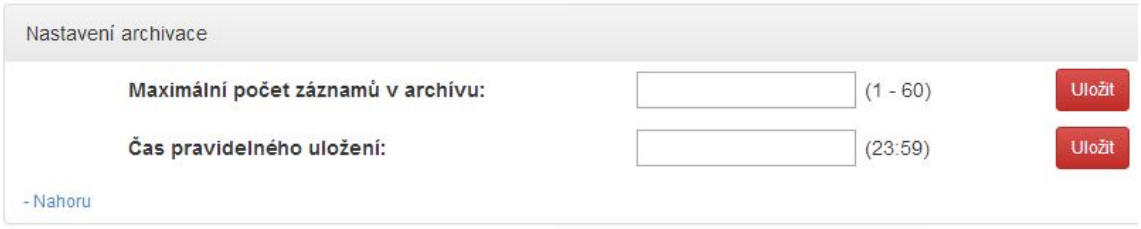
ITrigger je základní interface pro všechny Quartz.NET triggery. TriggerBuilder je metoda, která slouží k vytvoření instance ITriggeru. Dále vidíme metodu WithIdentity. Tato metoda slouží k identifikaci triggeru za použití klíče, který obsahuje jméno triggeru a skupinu. Poté už zbývá jen nastavit pravidla, podle kterých se trigger aktivuje. Tímto je vytvořen trigger, na který stačí zavolat metodu pro start ScheduleJobu a není třeba se dále o cokoliv starat. Jak je vidět, tato knihovna je i přes svou velkou funkcionalitu a možnosti tvorby komplexních úloh dobře pochopitelná a lehce se s ní pracuje.

4.4 Nastavení

V průběhu práce na tomto projektu postupem času vzniklo několik dalších požadavků na manuální úpravy aplikace. Patří mezi ně tyto:

- Obecné nastavení,
- editace beden,
- editace palet,
- editace konkrétních zákazníků.

V obecném nastavení se nyní vyskytuje dvojice předvoleb. Jde především o výše zmiňovaný maximální počet uložených Excel exportů, dále maximální výška, do které algoritmus vypočítá palety.



Obrázek 8: Vizualizace nastavení archivace

Nastavení palet a beden jsou dvě velmi podobné služby. Obě služby pomocí GridView control vypíší uživateli na obrazovku tabulku, ve které jsou přehledně vypsány všechny informace týkající se jak palet, tak beden. Jde především o ID, jméno, rozměry (šířka, délka, výška) a zda jsou stohovatelné. Díky potřebě zamezit vstupu do databáze je možné u obou výpisů přímo z webové stránky přidávat, upravovat či případně mazat záznamy. Všechny zmíněné úkony mají pro tyto účely v .NET frameworku své metody, které poskytnou základní funkčnost. Jedná se o metody OnRowCommand, OnRowEditing, OnRowUpdating, OnRowDeleting. Pro základní úkony s GridView elementem stačí, nicméně jsme se opět rozhodli funkčnost mírně obohatit a zlepšit tak uživatelský zážitek z aplikace tím, že pro editaci tabulek budeme využívat .NET AJAX CRUD, což je zkratka pro Create, Read, Update a Delete a navíc přidáme malý jQuery BlockUI plugin, který zabraňuje načítání UI aplikace do doby, než je provedena konkrétní operace. jQUery skript nicméně nebude blokovat kompletní UI aplikace, ale pouze její část, kde je umístěn GridView element. Díky tomu výrazně zrychlíme provedení celé operace a není třeba opět načíst celou webovou stránku.

K tomu, abychom mohli tabulky začít editovat je potřeba nejprve správně nastavit SQL server connectionString, což je struktura, která nám později umožní propojit GridView s databází. Každá databáze má svůj ConnectionString jasně definovaný a je tedy třeba jej do kódu přesně napsat. Poté co máme takto definovaný ConnectionString, můžeme se začít věnovat konkrétním metodám starající se o GridView elementy. První z

těchto metod je metoda BindData(). V této metodě pomocí řetězce zavoláme select příkaz z databáze a data propíšeme do GridView elementu. Select pomocí stringu vypadá následovně :

```
public class StringExample()
{
    String  exampleQuery = "Select␣example1,␣example2,␣example3" + "from␣exampleTable";
}
```

Výpis 4: Příklad selectu dat z tabulky exampleTable

Následuje metoda GetData, ve které nejprve načteme data z databáze a poté daty naplníme datovou strukturu DataTable, se kterou dále pracujeme. Jako první je nutné vytvořit instanci datové struktury DataTable. Poté si díky SQLConnection otevřeme spojení s databází a inicializujeme předchozí vytvořenou instanci ConnectionStringu. Dále je nutné vytvořit SqlDataAdapter, díky kterému pak budeme schopni naplnit strukturu DataTable(). Pomocí konstruktoru si vytvoříme instanci SQL adaptéru. Jako poslední důležitou metodu využijeme Fill(). Touto metodou naplníme DataTable. V tomto momentě máme načtené data z databáze a můžeme přistoupit k jednotlivým metodám editace.

První na řadě je metoda AddNewRecord(), která již podle jména značí, co vykonává. Jde o přidání nového záznamu do databáze. Na webu je umístěna jako poslední řádek samotného výpisu z GridView a zapadá do kontextu. Tělo metody se skládá z tří hlavních částí. Jde o napárování GridView control s textboxy, dále insert příkaz pro vložení nového objektu a hned za ním následující select příkaz pro okamžité zobrazení nově zadaných hodnot a jako poslední přiřazení hodnot zpět tabulce. Příklad pro vložení nového ID bedny:

```
public class AddValue()
{
    string BoxID = ((TextBox)GridView2.FooterRow.FindControl("txtBoxID")).Text;
    cmd.CommandText = "insert␣into␣Bedna(boxID)" + "␣values␣(@boxID)␣+␣"select boxID from
        Bedna";
␣␣␣cmd.Parameters.Add("@boxID",␣SqlDbType.VarChar).Value␣=␣BoxID;
}
```

Výpis 5: Příklad vložení nových dat do tabulky po manipulaci z klientské strany

Druhá metoda pro editaci GridView elementu přímo z klientské strany je DeleteRecord(). Opět je z názvu patrné, co je náplní práce této metody. Jde pouze o smazání záznamu. Nicméně oproti metodě, která vytvoří nový záznam je zde několik rozdílů. Z pohledu UI (user interface) zde vzniklo vyskakovací okno s potvrzením o smazání záznamu. Přidáno bylo jako obrana proti náhlému smazání, například vinou nechtěného kliknutí. Z technického pohledu je funkcionalita smazání záznamu v podobě tlačítka „smazat“ zabudovaná přímo v tabulce jako další sloupec. Díky tomu nijak neruší a je vždy jasné, co bude výsledkem této akce. Když uživatel klikne na „smazat“, otevře se vyskakovací okno s dotazem „Opravdu si přejete smazat tento záznam?“. Pokud uživatel klikne na ano, potvrdí tím smazání záznamu a záznam je vymazán z databáze a aktualizovaný GridView control na webové stránce opět hned vyobrazený včetně poslední

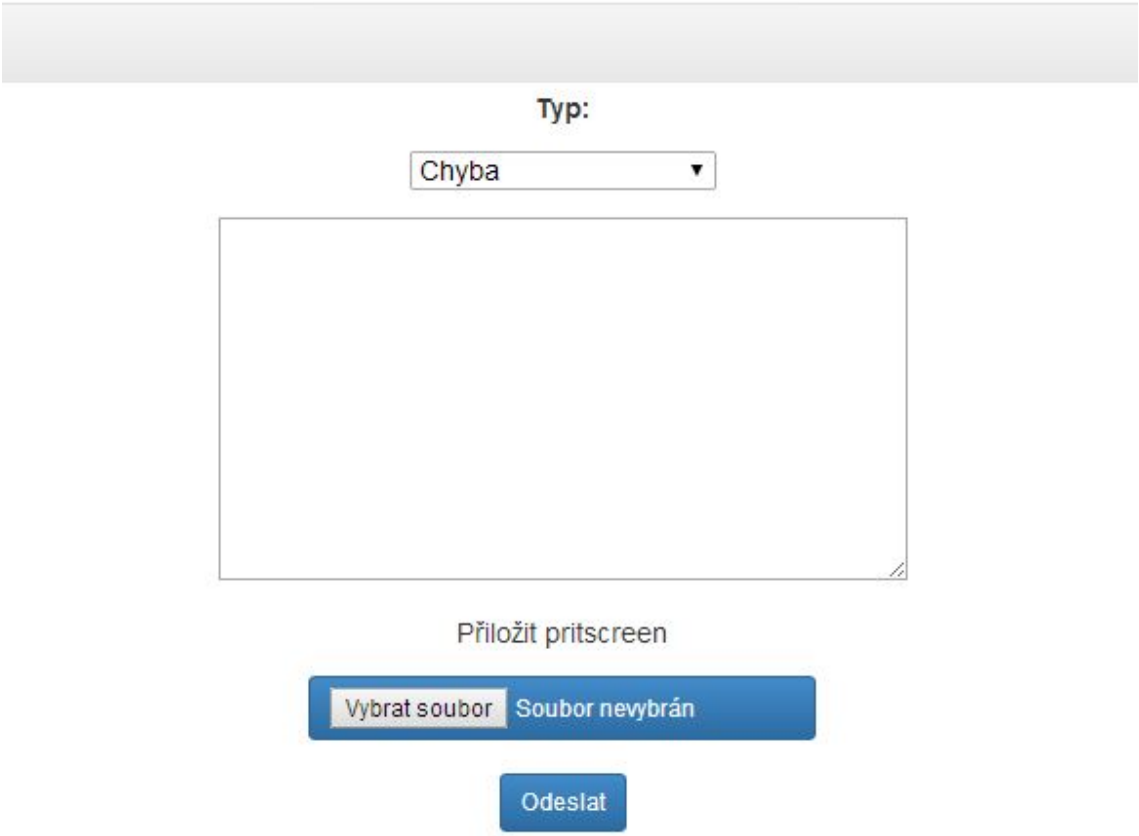
změny – tedy bez vymazaného záznamu. Na pozadí metody jako první vytvoříme instanci hyper-linkového tlačítka. Poté otevřeme spojení přes SQL connection a při otevřeném spojení díky metodě CommandText vložíme příkaz delete pro určitý řádek tabulky, kde se orientujeme podle primárního klíče, kterým je boxID. Hned poté přiřadíme druhý příkaz, kterým vybereme vše z nově upravené tabulky. Použijeme metody DataBind() a nově načtené data propíšeme do GridView control na klientské straně.

Proces pro editaci záznamu se skládá ze tří metod. Jsou jimi EditRecord(), CancelEdit() a UpdateRecord(). První metoda EditRecord() obsahuje pouze dva řádky. Využíváme zde metody EditIndex, která nám vrátí číslo řádku, který chceme editovat. Poté jen zavoláme BindData() a obnovíme tabulku. Nyní přichází na řadu druhá metoda - metoda UpdateRecord(). Je téměř identická jako naše první metoda AddRecord(). Jediným hlavním rozdílem je příkaz zasílaný do databáze. Zde se nejedná o INSERT command, nýbrž o UPDATE command. Druhá věc, která tuto metodu odlišuje, je udržování stavu řádku, který upravujeme. Pro tyto účely používáme metodu EditIndex. Tato metoda na začátku nastaví index řádku, který editujeme a na konci jej taktéž vrátí. Tím máme zajištěno, že vždy budeme upravovat právě jeden a ten samý řádek. Pokud se dostaneme do stavu, kdy již editujeme konkrétní řádek, ale z nějakého důvodu si to rozmyslíme, řešíme to třetí metodou – CancelEdit(). Tato metoda je velmi jednoduchá. Pokud je zavolána, stará se pouze o vrácení indexu řádku a zrušení nového zápisu. Jako poslední se provede opět BindData(). Díky tomu jsou data opět hned aktuální.

4.5 Kontakt

Poslední části intranetové aplikace pro firmu Gates Hydraulics s.r.o je kontaktní formulář pro rychlé zadání chyby či požadavku. Formulář obsahuje rozevírací seznam, ve kterém je možné rychle vybrat jednu z možností týkající se plánovaného sdělení. Jedná se například o kategorie chyba, požadavek na funkci, změna UI a podobně. Dále obsahuje textovou oblast sloužící k popsání sdělení. Pro lepší vizualizaci a vysvětlení problematiky formulář obsahuje také tlačítko pro přidání přílohy. Celý formulář je zakončen tlačítkem pro odeslání celé zprávy na námi zvolenou emailovou adresu.

Pro potřeby odeslání emailové zprávy slouží v C# prostředí systémová třída Net.Mail. Tuto třídu zde využíváme. Jako první vytvoříme instanci MailMessage. Na tuto instanci následně navážeme několik metod. Jedná se o metody From (od koho je zpráva zasílána, To (komu je zpráva zasílána), Subject (předmět zprávy – pro naše potřeby staticky zvoleno „Gates Hydraulics“) a metoda Body (samotné tělo zprávy). Díky možnosti přiložit přílohu je třeba vytvořit instanci objektu Attachement a namapovat ji na uživatelem zvolený soubor. Důležitou součástí fungování emailové komunikace je vytvoření instance SMTPClient. Při tvorbě instance rovnou sdělíme, jaký SMTP server budeme využívat a také jeho port. Pak už stačí jen zprávu odeslat metodou Send(mailMessage). Pokud se uživateli zpráva podaří bez problému odeslat, systém zobrazí zprávu s poděkováním. V opačném případě zobrazuje chybovou hlášku a vyzve k opětovnému pokusu za pár minut. Zajímavostí v řešení takovýchto formulářů často bývá validace správně zadané emailové adresy. Ta je řešena pomocí řetězce regulérního výrazu. Pro zajímavost aktu-



Obrázek 9: Vizualizace kontaktního formuláře

ální standardní regulérní výraz k prověření správnosti zadané emailové adresy vypadá takto (standard RFC 5322):

```
[a-z0-9!#$\%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$\%&'*+/=?^_`{|}~-]+)*@
(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?
```

Výpis 6: Příklad regulerního výrazu pro ověření správnosti emailové adresy

Takto napsaný regulérní výraz pro validaci emailové adresy funguje až na 99% veškerých dnes založených emailových adres.

4.6 Syntetické testy

Součástí této práce je i porovnání výsledků různých přístupů algoritmů k problému, podíváme se především na 2D a 3D varianty Bin Packing problému. V současnosti, z pohledu rychlosti není třeba efektivitu 2D Bin Packing algoritmu nijak výrazně měřit. Veškeré procesy čítající stovky kusů objektů se ve 2D formě počítají v řádu milisekund.

Horší je to s jejich účinností, kterou změřil a popsal ve své vyčerpávající studii Jukka Jylanki [10]. Z této studie plyne, že Shelf a Skyline algoritmy by měli být využívány pouze se značnými vylepšeními, protože veškeré algoritmy, které v reálné situace neudržují přehled o volné ploše kontejneru se nevyplatí používat. I když měl Skyline algoritmus mnohem lepší výsledky než Shelf algoritmus a dokonce o něco lepší výsledky než Guillotine algoritmus, díky výše zmíněnému se nevyplatí s ním pracovat. Jako nejlepší varianta pro řešení 2D verze byl s ohledem na rychlost a přesnost vybrán Maximal Rectangles algoritmus, který se blížil hranici 95%.

Typ algoritmu	Přesnost	Poznámka
Shelf algoritmus	62% (s optimalizací 81.20%)	nedoporučuje se
Guillotine algoritmus	90,95%	
Maximal Rectangle Alg.	90,92% (s optimalizací 94,06%)	
Skyline Algoritmus	91,61 %	nedoporučuje se

Tabulka 1: Tabulka efektivity výpočtů 2D Bin Packingu různými přístupy, Zdroj: [10]

Ve 3D verzi Bin Packing problému je to již složitější. Časová náročnost této verze je náročnější a rozdíly ve výsledcích mohou být taktéž výrazné. Samotné výsledky jsou velmi závislé i na tom, jak do programu vstupují objekty (bedna) a samozřejmě jakých algoritmů a upřesňujících podmínek se využívá pro výpočet. Nelze proto úplně říci, který algoritmus by byl nejlepší. Vždy totiž záleží na konkrétní situaci a požadavcích klienta na sestavení algoritmu a podmínek na míru. Pokud bychom měli srovnat 3D Bin Packing algoritmy prezentované v této práci, zjistíme, že Next-Fit algoritmus je nejrychlejší, protože se vždy stará jen o jednu bednu. Naopak First-Fit algoritmus je celkově jeden z nejpomalejších, protože prochází celou strukturou neustále dokola. Podrobně analyzoval metody Bin Packing problemu také Bastian Rieck [13] Mimo jiné se techniky výpočtu nákladů neustále vylepšují, což dokazuje například nedávný výzkum nové metodiky Bin Packing problému, zvaný Peek Filling Slice Push (PFSP) [12]. Podle výzkumu tato metoda překonává starší metody a zrychluje čas potřebný pro dosažení výsledků.

5 Vizualizace

Během vývoje systému pro firmu Gates s.r.o. se nakonec ukázalo, že pro potřeby firmy není vizualizace výsledků žádoucí. Ve skladu není prostor na sledování vizualizovaného výsledku, a proto byl tento prvotní požadavek vypuštěn. Nicméně pro potřeby splnění zadání mé práce zde uvedu a rozvedu pár možností, kterými bych se v případě realizace zabýval. Jedná se především o frameworky určené svými možnostmi pro herní průmysl. Konkrétně se jedná o XNA Framework od firmy Microsoft a také Unity Engine ve své verzi 4 vyvíjený firmou Unity Technologies. XNA Framework je méně obsáhlý Framework a neposkytuje takové možnosti. Naproti tomu Unity Engine, je komplexní herní vývojové prostředí, umožňující vývoj těch nejkvalitnějších her. Možnosti Unity Enginu dalece přesahují naše požadavky, nicméně jsme schopni v něm tyto vizualizace jednoduše provádět. Výhodou obou enginů je také fakt, že jako programovací jazyk využívají C#. V případě Unity Enginu ještě také Javascript. Není tedy třeba učit se kompletně novému jazyku, ale pouze věcem, které jsou specifické pro daný Framework, či engine.

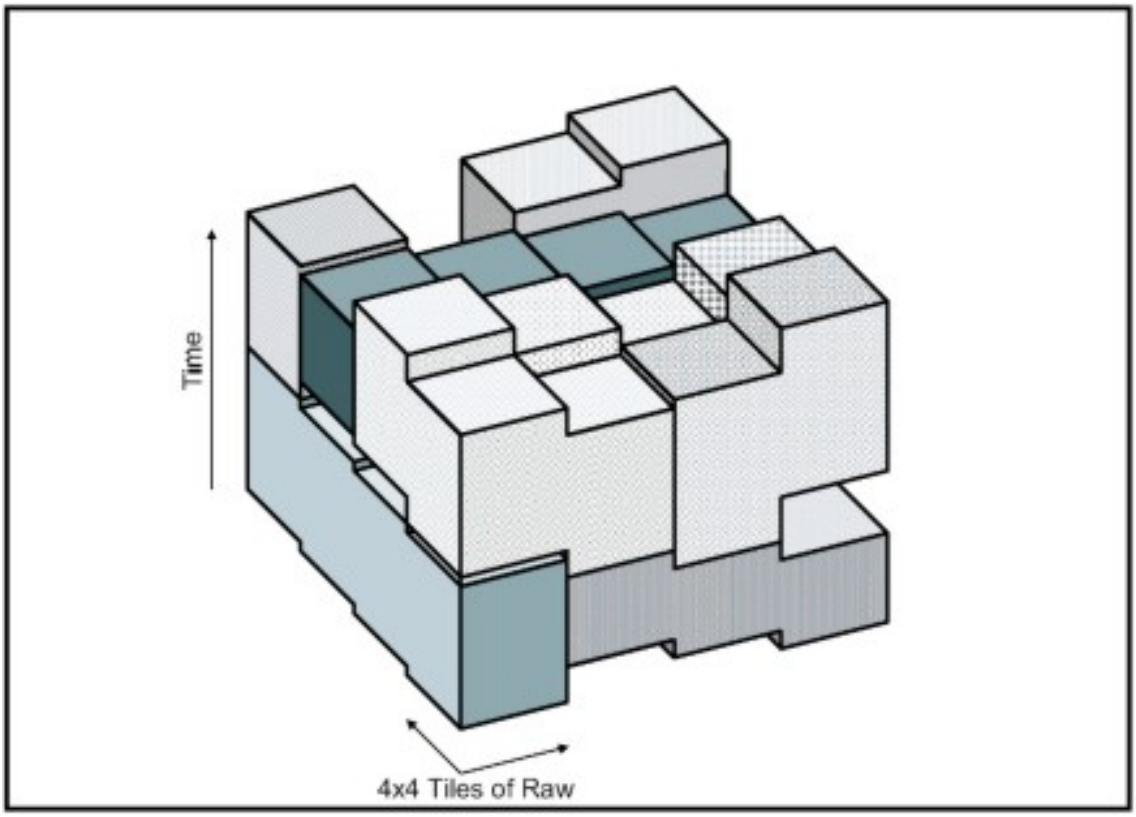
5.1 XNA Framework

XNA Framework je sada knihoven do jazyka C#. Vývoj pomocí tohoto frameworku probíhá v SDK od firmy Microsoft, kterým je Visual Studio. XNA Framework vznikl jako vysokoúrovňová nástavba nad DirectX pro C# .NET, určená pro pohodlnou tvorbu grafických aplikací a profesionálních her. Problémem DirectX je, že práce s ním je problémová a většinou realizovaná pouze přes jazyk jako je například C++, což je nízkoúrovňový jazyk s manuální správou paměti. Díky tomu se často stává, že přímý vývoj na DirectX je těžkopádný a složitý. Po příchodu jazyka C# přišla firma Microsoft také s novou verzi DirectX, kterou pojmenovala Managed DirectX. Hlavní změnou v této nové verzi byla automatická správa paměti. Tato nová verze DirectX využívá až 98% výkonu klasického DirectX a zároveň také umožňuje vysokoúrovňovou práci, v našem případě třeba v jazyce C#. Z Managed DirectX nakonec vznikl XNA Framework. Aktuálně je XNA Framework pohřben. Firma Microsoft jej odstranila ze svého portfolia aktivních služeb a nadále jen oficiálně nepodporuje i přes své zjevné kvality.

5.2 Unity Engine

Unity Engine je robustní vývojové prostředí pro tvorbu moderních herních titulů té nejvyšší kvality na všechny moderní platformy současnosti. Jedná se především o PC, XBOX, PS4, iOS, Android a Windows Phone, případně webové rozhraní. Je velmi komplexní a obsahuje nepřeberné množství možností, navíc je v základní verzi zcela zdarma. Vývoj v Unity Enginu je velmi flexibilní a dá se proto použít i pro naše potřeby, tzv. pouhá vizualizace řešení ve webové aplikaci. K tomu poslouží Unity Web Player plugin, které byl spuštěn v roce 2005 jako další větev vývoje tohoto enginu. V roce 2014 je to již velmi dobře podporovaná platforma, která poskytuje jak vývojářům, tak hostiteli multimediálního obsahu velmi dobré možnosti vývoje 2D a 3D interaktivního obsahu na webu. Poté co je vyvinuta požadovaná funkcionalita, v Unity stačí exportovat veškerou práci pro

Unity Web Plugin a Unity Engine sám vygeneruje téměř všechny potřebné HTML kódy pro spuštění na webové stránce.



Obrázek 10: Ukázka vizualizace 3D Bin Packingu, Zdroj:[11]

6 Závěr

Jsem rád, že jsem se mohl účastnit procesu tvorby tohoto systému pro firmu Gates Hydraulics s.r.o. Za dobu práce na tomto projektu jsem se naučil mnoho nového, vyzkoušel několik neznámých věcí a hledal řešení na problémy, které jsem nikdy předtím neřešil. Bylo to velmi zajímavé a jsem rád, že jsem se do této bakalářské práce pustil. Myslím, že se mi podařilo vytvořit systém, který přesně kopíruje požadavky firmy Gates Hydraulics s.r.o. Společně s kolegou Bc. Janem Dittrichem, který zodpovídal za vizuální stránku věci, jsme pak vytvořili funkční a moderně vypadající systém pracující na intranetu společnosti. Doufám, že projekt Packing-Manager bude pro firmu Gates Hydraulics s.r.o. v dlouhodobém horizontu úspěšný a pomůže v dalším rozvoji firmy.

Z pohledu budoucího vývoje tohoto projektu je zde více možností. První z nich je dále pokračovat ve spolupráci s firmou Gates Hydraulics s.r.o a snažit se zdokonalit algoritmus pro jejich potřeby. Druhou možností je mít obecnou verzi algoritmu, na kterém se dá dále pracovat a hledat metody, jak zpřesnit a zrychlit metodiku výpočtu. Poslední možností je komerční využití projektu. Služby tohoto typu jsou čím dál více žádané a přepravní společnosti v České Republice začínají zjišťovat, že i k účelům optimalizace nákladu se dá využít počítačů s následnou velmi rychlou návratností investice.

Jakub Bílý

7 Reference

[1] *Gates Corporation* http://www.gateshydraulics.cz/

[2] V.S. Tanaev, V.S. Gordon, Y.M. Shafransky, *NP-Hard Problems* Mathematics and Its Applications Volume 284, 253-311, 1994.

[3] S. Walukiewicz, *Knapsack Problem and its Generalizations*, 1985.

[4] S. Martello, D. Pisinger, D. Vigo *The Three-Dimensional Bin Packing Problem*, 1997.

[5] Stanford University *The Tree Data Model*, http://infolab.stanford.edu/ ullman/focs/ch05.pdf 2010.

[6] D .Pisinger, *Algorithms for Knapsack Problems* http://www.diku.dk/ pisinger/95-1.pdf 1995.

[7] A.L. Corcoran III, R.L. Wainwright, *A Genetic Algorithm for Packing in Three Dimensions* The University of Tulsa 1992.

[8] E.E. Bischoff, M.D. Marriott, *A comparative evaluation of heuristics for container loading*. European Journal of Operational Research, 44:267-276, 1990.

[9] J.A.George, D.F. Robinson, *A heuristic for packing boxes into a container* European Journal of Operational Research, 7:147-156, 1980.

[10] J. Jylanki, *A thousand Ways to Pack the Bin - A Practical approach to Two-Dimensional Rectangle Bin Packing*, 2010.

[11] M. Gordon, S. Amarasinghe, *Space - Time Multiplexing of Stream Programs*, http://publications.csail.mit.edu/abstracts/abstracts06/mgordon/mgordon.html 2006.

[12] W. F. Maarouf, A. M. Barbar, M. J. Owayjan, *A New Heuristic Algorithm for the 3D Bin Packing Problem* 2008.

[13] B. Rieck, *Basic Analysis of Bin-Packing Heuristics* 2009.

[14] *EPPlus-Create advanced Excel 2007 spreadsheets on the server* http://epplus.codeplex.com/

[15] *Quartz.NET is a full-featured, open source job scheduling system* http://www.quartz-scheduler.net/

[16] R. Mall, *Fundamentals of Software Engineering, 2nd ed*. ISBN-81-203-2445-5, 2003.

A Přílohy

Práce obsahuje dvě přílohy ve formě DVD, které budou přiloženy v pevných deskách.

První DVD je veřejné, lze na něm najít třídní diagramy, use case diagramy a také obrázky z finální verze aplikace pro firmu Gates Hydraulics s.r.o.

Druhé DVD je neveřejné a obsahuje zdrojové kódy celé aplikace. Toto DVD bude poskytnuto na vyžádání a pouze na nezbytně dlouhou dobu pro oponenta této bakalářské práce nebo komisi.